Calhoun

Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations 1. Thesis and Dissertation Collection, all items

1973

Microprogrammed disc controllers.

Tomlin, Edwin Ladeau.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/16701

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

MICROPROGRAMMED DISC CONTROLLERS

Edwin Ladeau Tomlin

Library
Naval Postgraduate School
Monterey, California 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MICROPROGRAMMED DISC CONTROLLERS

by

Edwin Ladeau Tomlin, Jr.

Thesis Advisor: V. M. Powers

December 1973

T157087

Approved for public release; distribution unlimited.

Microprogrammed Disc Controllers

by

Edwin Ladeau Tomlin, Jr.
Ensign, United States Navy
B.S., United States Naval Academy, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1973

The.
F727
c.1

Library
Naval Postgraduate School
Monterey, California 93940

ABSTRACT

Disc controllers, the concepts of microprogramming, and the combination of the two are presented. First, the conventional controller is discussed and a model developed. Logic for the model is then replaced by microprogramming to form a new model. Finally, the new version is modified to handle extended logic. The functions, structure, and relative merits of the three models are discussed, as well as suggestions for further work.

TABLE OF CONTENTS

I. INTRODUCTION

The reasons for development of smart controllers for peripheral devices are extensive, and the advantages of microprogramming are many. The combination of these two facets of the computer field presents interesting and powerful possibilities. Thus, the purpose of the following investigation was to examine the two areas separately and collectively and to draw conclusions about their feasibility and potential for use in future computer systems.

The peripheral device chosen for investigation was the disc drive due to availability of abundant and detailed information on such systems. A specific disc drive and controller system was chosen and was simplified to form a working model. Microprogramming logic was then substituted for the conventional logic of the system, but only to carry out the same functions. Finally, the microprogrammed logic was extended to form a "smart" controller capable of doing more powerful functions.

The investigation and results are presented here. Also, possible extensions of this study and directions for development in related topics are suggested.

II. NATURE OF THE PROBLEM

A. DISC CONTROLLERS

In order to extend the capabilities of a computer, any of several types of peripheral devices may be added to supplement the system. However, each of these devices requires additional equipment, known as a controller, to perform the proper interfacing functions. Controllers coordinate input and output between the main computer and one or more peripheral devices.

Attention here is focused on a specific peripheral device, the disc drive, and the controller associated with it. Unfortunately, complete general analysis of such a system tends to become bogged down in trivialities. Therefore, for purposes of this paper, the disc controller examined will be a simplified model of an existing system — the ADAGE Disc Memory Subsystem, or DMS2 [Ref. 1]. The disc drive is the hardware which contains the data storage medium, or discs. In the system being examined there are two discs, of which each is divided into 203 concentric rings known as cylinders. The cylinders are further divided into eight equally-sized sectors, which are the smallest addressable units of storage. The disc controller, upon request from the main computer, should be able to do at least the following functions: select a drive, seek and find a cylinder, write pertinent identification information at the head of every sector of a specified cylinder, read data from the disc drive

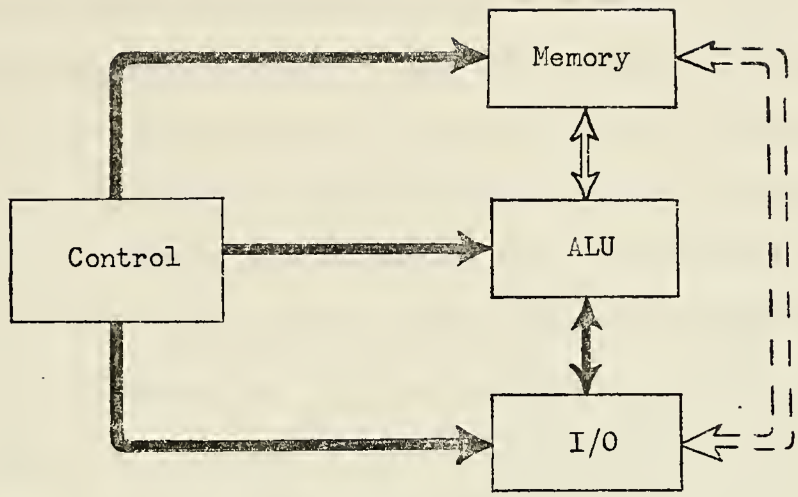
and write data on the disc drive. In order to carry out these functions, an extensive number of registers and timing circuits must be manipulated by the included logic.

B. MICROPROGRAMMING

In recent years the concept of microprogramming has found extensive application in computer logic design. It has proved popular for several reasons, which can best be understood by comparison of conventional and microprogrammed control. The block diagrams in Figure 1 show the four basic sections of a computer for both the conventional and microprogrammed versions [Ref. 2]. Logic circuits in the conventional computer are distributed throughout the machine; moreover, the logic includes extensive timing signals which subdivide the machine cycle so that control actions will occur in the proper sequence. While efficient, this method is both complex and inflexible. Even small changes require difficult analysis and redesign.

Microprogrammed control, on the other hand, is very flexible, and additions or deletions to such machines are straightforward. Machine control is centralized in the Control Store portion of the computer, as represented in the diagram. The hardware here may be Read Only Memory (ROM), or Random Access Memory (RAM) which allows both reading and writing at the expense of speed and cost, or it may be a type of storage element with some properties of both. Within the Control Store section are several microprogarms; each microprogram controls, typically, one machine instructions. A

CONVENTIONAL CONTROL



MICROPROGRAMMED CONTROL

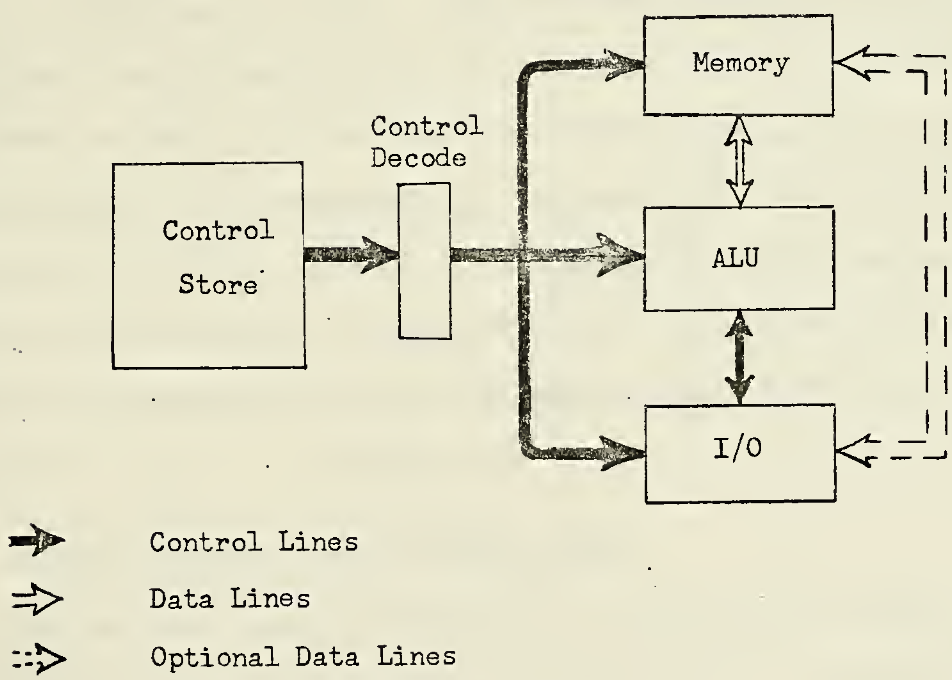


Figure 1.

microprogram consists of several microinstructions, and each microinstruction can be broken down into several fields which contain the micro-orders. The micro-orders are decoded in the Control Decode section to form the control signals to the rest of the computer. Control Decode is also responsible for fetching the correct microprogram from the Control Store after receiving the machine code from the I/O section [Ref. 2].

Many of the benefits of microprogramming should now be obvious. First, the timing of most signals is automatically taken care of by the fact that only one microinstruction at a time can be executed. Also, flexibility of the instruction set is of major importance. Third, microprogramming is certainly a more orderly approach to control section design, and, therefore, errors are fewer and easier to correct. A fourth advantage is cost. The microprogrammed computer requires fewer components and consists mainly of the relatively inexpensive ROM. Since there are fewer parts, there are fewer malfunctions, and reliability is increased. The only major drawback of microprogramming is its performance relative to conventional control. Microprogrammed logic is generally slower, and some of the more sophisticated features of large computers cannot be implemented.

C. MICROPROGRAMMED DISC CONTROLLERS

For several reasons controllers traditionally have been fabricated with conventional control logic. Historically, microprogramming did not become cost effective until recently, when the cost of ROMs was reduced sufficiently. Also, many

of the advantages afforded by microprogramming did not apply to the dedicated logic of the controller. For example, microprogramming allows flexibility in implementing new functions, but the designer of peripheral control equipment has up to now been concerned only with building a specialized piece of hardware for a dedicated task. Extension of the function set which the device would handle was not considered necessary or practical, and it is not hard to understand why that should be true. Controllers have always been considered to be only interfaces, and not machines which do any sort of intelligent data handling; any manipulation of information was done by the computer. Thus, a more powerful controller simply meant a more efficient way to do a brute force task, and efforts in that direction were not worthwhile.

On the other hand, "smart" controllers are certainly feasible and function sets are a prime consideration in such devices. This can best be seen by returning to the example of the disc controller. A typical such unit (somewhat simplified) may be able to handle the following functions: Select, Seek, Write Format, Read, Write, and Sense Status. On the other hand, a "smart" disc controller may be able to handle the following: Seek-Select, Read, Write, Sense Status, Write Format All (an entire disc), Insert Record, Delete Record, Collect Garbage, Alphabetize, List, and any number of functions to handle the data to be read or written. All such functions could be done without involving the central computer. Such a controller, then, would certainly save on

CPU time, which is the prime objective of such a device. Furthermore, since the functions are contained in the ROM, it is both practical and convenient to change the functions as the application changes. In order to implement a different set of functions, one would simply replace the plug-in ROM with the proper module or cause the CPU to electronically switch to a proper section of ROM.

In light of the above, a microprogrammed disc controller becomes more practical. Because the functions desired to be performed may change, flexibility is certainly a benefit. Likewise, the other advantages apply as well, but in this application it is the flexibility due to the modularity of the functions which makes microprogramming superior to conventional control logic design.

III. TYPICAL CONVENTIONAL CONTROLLER

In order to access data stored on a disc, there are certain functions which a conventional controller must be able to execute. The logic to carry out even these basic functions requires an extensive array of registers and logic elements. Since complete analysis of such an existing system is both tedious and complicated, simplification is necessary. The simplified model used for reference here was derived from the ADAGE DMS2 [Ref. 1]. It attempts to show sufficient detail of such a system, emphasizing the necessary elements without oversimplifying. The logic components mentioned in part A below are described in part B.

A. FUNCTIONS AND THEIR EXECUTIONS

The central computer can send one of six commands to the disc controller to perform various functions. The Select command designates a specific drive with which the main computer wishes to communicate. The necessary lines to that drive are enabled, and the ones to other drives inhibited, until another Select command is encountered. One disc controller can handle one of up to four drives at any one time.

The Seek command specifies the number of cylinders to move and the direction of the seek. First, the number of cylinders to move is subtracted from 255 and the seek head is set in motion either forward or reverse. As the head

passes over cylinders on the disc, incrementing pulses are sent to the Cylinder Step Counter (CSC) Register, which monitors the distance left to move. When the CSC reaches 235, the Seek Slow Flip-Flop is set, and it continues to be set until the CSC is all ones (255). The Seek Stop Flip-Flop is set and the seek is complete.

The Write Format command causes the controller to write header information on all sectors within the current cylinder. First, the controller waits for the arrival of the index pulse, which occurs just before Sector 0; it then waits 95 bit times before writing. The 95 spaces are known as a header gap, which is followed by the Sync Bit. Then four bytes of information are written after the Sync Bit:- track number, cylinder number, sector number, and cyclic check bytes. Following this header information is another header gap, which is again concluded with a Sync Bit, and the controller is ready for the next sector. Sector pulses, which are equivalent to index pulses at sectors other than zero, initiate the Write Format cycle for the remaining sectors until all sixteen headers have been written (eight sectors on two tracks constitute a cylinder).

Upon receipt of a Read command, the controller waits for a sector pulse to arrive. After its arrival, the controller then enables the read gate and waits for the Sync Bit. The header information on the sector being read is comared bit-by-bit to that of the desired sector, and if the two are not identical, various events may occur. A track conflict causes

the head to switch tracks before reading the next header; a sector conflict causes the controller to wait for the next sector pulse; a cylinder or check byte conflict will cause error routines to be initiated — the appropriate bits being set in the error register. If the header information does agree with that of the desired sector, the read clock is enabled and the controller waits again for the Sync Bit. Data is read out serially into an eight-bit shift register and transferred a byte at a time, first to a data buffer and from there to the central computer. The Read command terminates on a signal from the computer indicating that the desired data has been read in.

A Write command is similar to a Read. The same header comparison routine is involved, and only after the correct sector is found do the two procedures differ. For the Write command, the read gate is inhibited and the write gate is enabled along with the write clock. The data flow is reversed, passing from the CPU into the buffer register a byte at a time and likewise into the shift register. From there it passes serially onto the disc. As before, the operation is terminated by a signal from the computer stating that all data has been passed to the controller.

The Sense Status command may be sent to the controller at any time, whether or not it is busy with another command. When the controller sees that a status report has been requested, it merely places the contents of the Status

Register on the bus to the computer. This operations is completely independent of any others which may be occurring.

B. STRUCTURE

A block diagram of the simplified model of a typical conventional disc controller is shown in Figure 2. Data transfers of more than one bit at a time are denoted by a wide line, whereas a single line indicates a one-bit or pulse transfer. The data link with the computer is via the sixteen-bit C-bus, which is connected to both the Device Command Receiver and Data Buffer. In addition, there are four signal lines linking the Device Command Receiver to the CPU; these are used in conjunction with the transfer of commands to the controller.

Almost all the logic for the device is contained in the Controller Logic and Device Command Receiver blocks. The latter controls the input into the Command Register and, in part, the output from the Status Register as well as the clock. Figure 3 shows the Device Command Receiver for this controller model (and subsequent ones as well). The Controller Logic block handles significantly more signals and variables. Included therein are the necessary timing circuits for proper sequencing of the steps of each command; decoding, signal routing, and other logic functions are grouped there also.

Controlled by the above logic are the numerous flip-flops and registers, each of which has a special task, as follows:

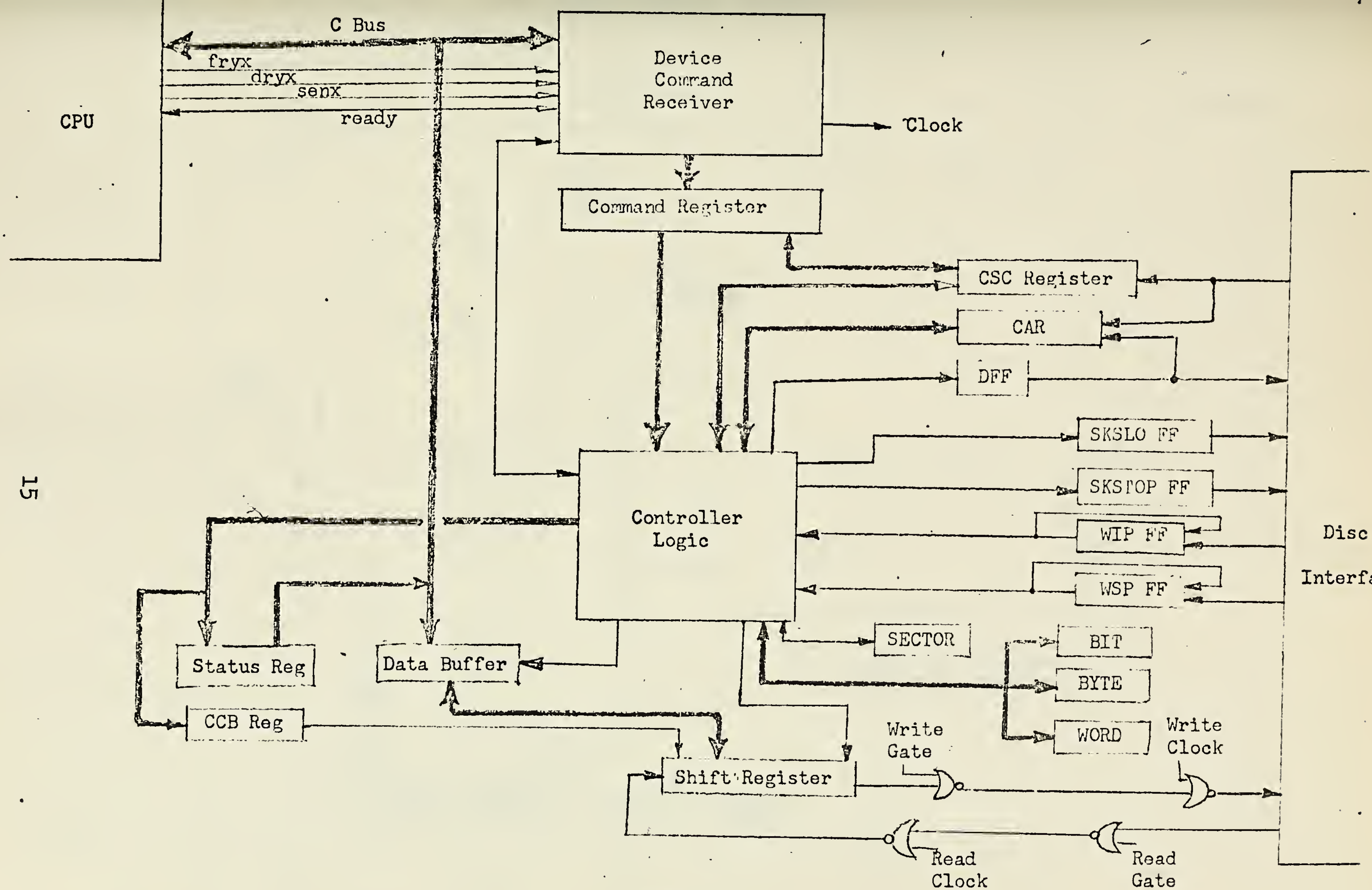

C Bus
CPU
fryx
dryx
senx
ready
Device Command Receiver
Clock
Command Register
CSC Register
CAR
DFF
SKSLO FF
SKSTOP FF
WIP FF
WSP FF
Controller Logic
SECTOR
BIT
BYTE
WORD
Status Reg
Data Buffer
CCB Reg
Shift Register
Write Gate
Write Clock
Read Clock
Read Gate
Disc
Interfa

Figure 2.

DEVICE COMMAND RECEIVER

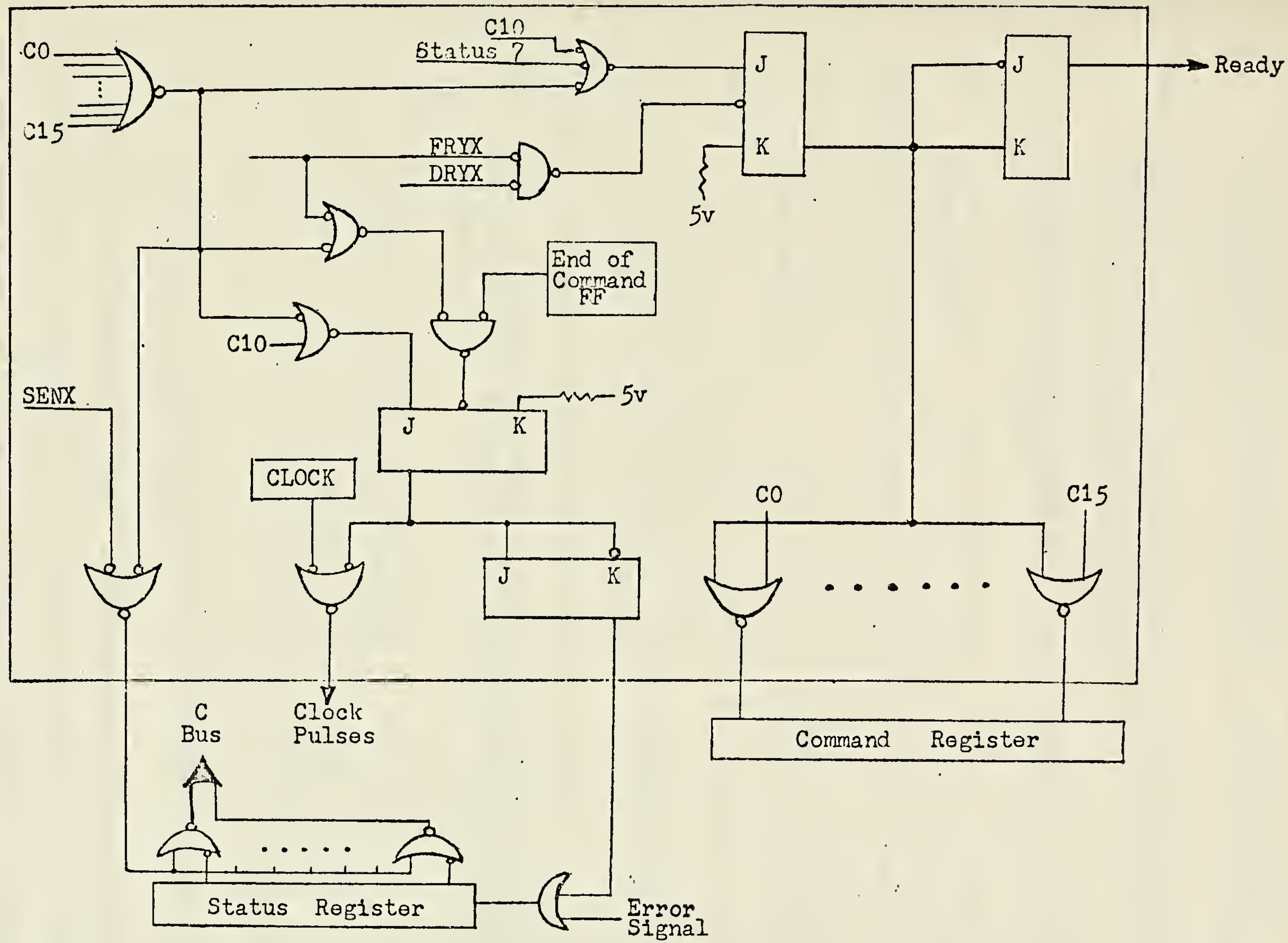


Figure 3.

Command Register — receives commands from CPU on the C-bus via the Device Command Receiver. Holds command throughout its execution for decoding and routing certain portions.

CSC Register — Cylinder Step Counter: Controller Logic monitors this register to set Seek Slow and Seek Stop Flip-Flops during Seek command.

CAR — Cylinder Address Register: keeps track of current track and cylinder.

DFF — Direction FF: seek forward when set, seek reverse when reset.

SKSLOFF — Seek Slow FF: when set, slows rate of seek of disc drive.

SKSTPFF — Seek Stop FF: when set, seek head is prevented from moving.

WIP FF — Wait for Index Pulse: set by index pulse, reset controlled by Controller Logic.

WSP FF — Wait for Sector Pulse: set by sector pulse, reset controlled by Controller Logic.

SECTOR — Keeps track of current sector number.

BIT, BYTE, and WORD Register — five-, two-, and two-bit counters used during Write Format, Read, and Write commands.

Shift Register — eight-bit register for holding data to be written and data read from disc.

CCB Register — Cyclic Check Byte register for parity checking.

Data Buffer — Intermediate register between Shift Register and CPU.

Status Register — Contains error bit, controller-busy bit, and six-bit error code.

The Status Register is very important to the operation of the controller. Setting of its controller-busy bit prevents the computer from interrupting a command which is in progress. However, at any time the CPU can request information with a Status Sense command, in which case the contents of the Status

Register are placed on the C-bus. The computer can then examine this information to determine whether the computer is busy and, if not, whether operation was termianted due to an error. If that is the case, the six-bit error code will pinpoint the problem, allowing the computer to take proper action.

C. SIMULATION

The purpose of the computer simulation of a model of a conventional disc control unit was to demonstrate the sequence of events that the controller carried out during each command. It was hoped that eventually the output from this simulation could be compared to the output of other simulations to amplify the similarities and differences. An effective investigative analysis of the controller's efficiency is probably not valid without significant extension of the simulation. That is, to obtain meaningful results as to whether or not the controller is doing an effective job would require monitoring more flip-flops and timing circuits. Such an extension for the program would not be difficult and would be a proper sequel to this investigation.

The program begins by assuming all discs are blank and no cylinders have headers written. Any of the commands except Sense Status may be performed. The commands to be simulated are read from individual data cards as a list of sixteen integers (0 or 1), which are equivalent to the machine codes for those commands. It should be mentioned that the data which are transferred during the Read and Write commands are

'pseudo-data'. That is, they are internally generated and are not stored for further use. Also, it was felt that parity checking was not necessary. This facet of the controller was handled by simply assigning zero to the cyclic check byte.

IV. MICROPROGRAMMED DISC CONTROLLER

The microprogrammed disc controller model described in this chapter is similar in many aspects to the non-microprogrammed version. Communications requirements with the CPU are identical in hardware and software. Non-decision-making hardware is the same, also. However, in this micro-programmed controller model, the control logic is carried out differently, as is communication within the device itself. This controller is neither faster nor more powerful, but it is more flexible, easier to design, and potentially less expensive.

A. FUNCTIONS AND THEIR EXECUTION

The six commands executed in the conventional controller are done here also. However, rather than hardwired logic, it is firmware that controls the operations. That is, the microprograms, which are stored in the ROM, regulate all necessary controller actions. They do this by defining all the required actions, which become micro-orders, then grouping them to form steps in the program for that function.

For purposes of this investigation, a 24-bit microprogram word size was adopted. The five fields were chosen large enough to accommodate the possible codes for each field listed in Appendix B. No significant effort was made to optimize the word size, field sizes, or binary encoding of the micro-orders.

All the microprograms in the ROM are listed in Appendix A, and one particular example is demonstrated here. In order to illustrate the process of executing a command, the Seek instruction is analyzed step-by-step to follow the logic scheme. After the command has been received by the Command Register and has been mapped into the proper ROM address (see Figure 4), the process begins. Every 196 nsec. the word at the indicated ROM address is loaded into the ROM Instruction Register, its different fields decoded, and then is executed. Control signals and data are carried on the 16-bit Internal (I) Bus.

ROM Address	Micro-Instruction
0010	[NOP D CR (20)] — Direction FF is set (or reset); the constant 20 is loaded into the A-Register.
0011	[C1 NOP SUB A NOP] — The contents of C1 (i.e. Command Register(8-15)), which contain the number of cylinders to move, is subtracted from the contents of the A-Register (20), and the result is placed in the A-Register.
0012	[OPP STP NOP NOP NEG] — If the result of 0011 is non-negative, then cylinders to move must be equal to or greater than twenty; in that case, skip next step (which leads to a seek slow routine).
0013	[NOP NOP JMP (020)] — If cylinders to move is less than twenty, this step will be executed. Jump to address 020 (seek slow routine).
0014	[A NOP RPT CSC NOP] — Contents of A-Register are loaded into Cylinder Step Counter. Seek Stop FF is reset, allowing seeking to begin. Repeat mode is set.
0015	[CSC NOP NOP A AZR] — Contents of Cylinder Step Counter (which is decremented each time disc drive passes over a track) is loaded into A-Register. This step repeats until A-Register, and, thus, CSC equals zero.

ROM Address	Micro-Instruction
0016	[NOP SL01 CR (20)] – Seek Slow FF is set. The constant 20 is loaded into A-Register, since there are twenty cylinders left to move.
0017	[A NOP RPT CSC UNC] – Contents of A-Register loaded into CSC. Repeat mode set. Next instruction is skipped unconditionally.
0020	[C1 SL01 RPT CSC NOP] – Follows instruction 0013. C1 (cylinders to move) is loaded into CSC. Seek Slow FF is set. Repeat mode set.
0021	[CSC NOP NOP A AZR] – Same as instruction 0015.
0022	[NOP SL02 NOP NOP EOP] – Seek Slow FF reset; Seek Stop FF set, thus inhibiting seek. EOP causes controller-busy bit in Status Register to be reset, thus ending the instruction.

B. STRUCTURE

A block diagram of the microprogrammed controller model is shown in Figure 4. Those registers having the same labels as the registers in the conventional controller also are identical physically and functionally. Data passed on the C-Bus and signal lines are also the same. The I-Bus is the path for internal routing of control and data information. The Controller Logic section has been replaced by a group of blocks which handle the microprogrammed logic, as follows:

MAPPER – decodes the first four bits of the command word to form an eight-bit ROM address which is subsequently loaded into the ROM Address Register.

ROMAR – ROM Address Register: holds the address of the microinstruction currently being executed. The address is incremented after execution of each microinstruction or modified according to the current instruction, (e.g. A Jump micro-order would load a specified address into the ROM Address Register.)

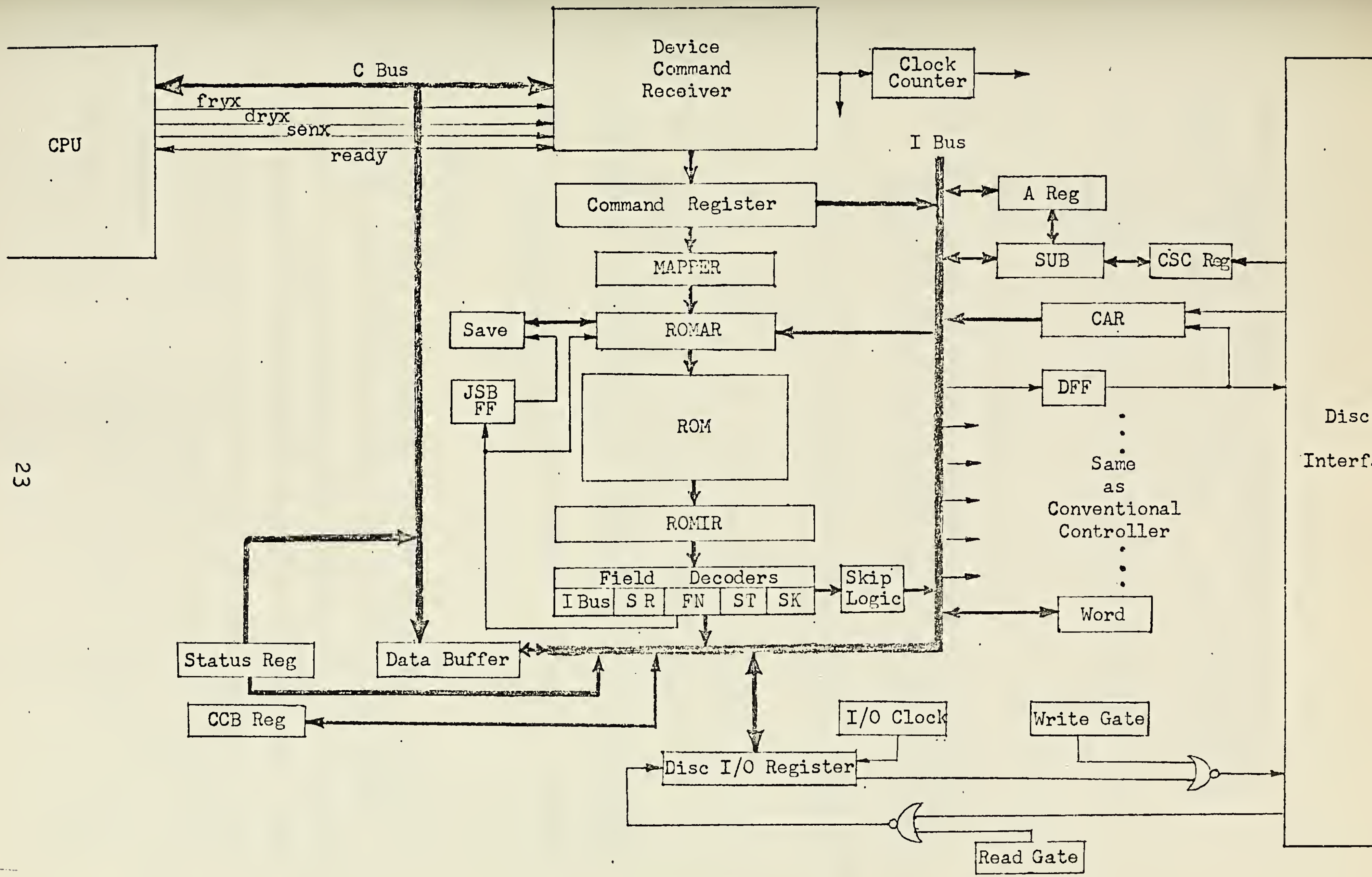

CPU
C Bus
fryx
dryx
senx
ready
Device Command Receiver
Clock Counter
I Bus
Command Register
A Reg
SUB
CSC Reg
MAPPER
Save
ROMAR
CAR
JSB FF
DFF
ROM
Disc Interf
Same as Conventional Controller
ROMIR
Field Decoders
I Bus
S R
FN
ST
SK
Skip Logic
Word
Status Reg
Data Buffer
CCB Reg
I/O Clock
Write Gate
Disc I/O Register
Read Gate

Figure 4.

ROM — Read Only Memory: contains the logic for the controller in the form of various microprograms.

ROMIR — ROM Instruction Register: contains the micro-instruction of the addressed memory location.

Field Decoders — decode the five fields of the micro-instruction in the ROM Instruction Register (i.e. the micro-orders) to form the control and data signals.

Skip Logic — logic necessary to test a skip condition and modify the ROM Address accordingly.

JSP FF and Save Register — Jump Subroutine Flip-Flop and Save Register are the logic necessary to hold the address of the microinstruction to which the program will return after executing a subroutine.

Clock Counter — steps down rate of clock from 196 nanoseconds cycle time to yield pulses every 784 nanoseconds. Both rates are used in the controller.

I/O Clock — combines the functions of the Read Clock and Write Clock in the conventional controller.

Disc I/O Register — equivalent functionally to Shift Register in the conventional controller.

A-Register — general purpose register used in conjunction with the subtractor.

SUB — Subtractor: used for necessary functions of subtraction and decrementing.

C. SIMULATION

The process of simulating a microprogrammed controller is significantly different from that of the conventional controller. In the microprogrammed simulation, all the micro-instructions are loaded into an array five rows wide. Each row contains a micro-order of the instruction except for the case in which bits zero through seven contain a constant or jump address. In that case, the Store and Skip fields of the array are blank and a specific assignment statement handles

the number. Whereas in the conventional controller each command had a different sequence of control logic, all commands in the microprogrammed version undergo the same basic logic check. Although all five fields would be acted upon simultaneously in the actual machine, the nature of the computer prevents simulating such action, and, thus, micro-orders are analyzed sequentially.

The command simulated for purposes of demonstration was the Seek command. (See Appendix C.) Only the logic necessary to that command's successful operation was implemented; however, extension to include all other commands is straightforward. First, the microinstructions in the form of literals are stored in an array which simulates the ROM. Then the Command Register is analyzed to determine the starting ROM Address; this simulates the Mapper. The words of the array then become the five micro-orders which are analyzed beginning in the I-bus field and moving on until all logic is completed. The address is updated and the program continues until an EOP terminates operation.

Overall, the program flow is much easier to follow than its conventional counterpart. While this is no proof as to their relative efficiencies, it does point up the micro-programmed controller's advantage in breaking down the logic to discrete groups of functions. Extensions of this simulation are easier and results are more significant than that of the conventional controller.

Although timing considerations do not affect the simulation, they should be noted here. In the model cycle time between microinstructions is 784 nanoseconds; this time period is broken down into four periods of 196 nanoseconds each. This is done because some microinstructions contain micro-orders which cannot be done simultaneously or must be done sequentially to properly perform the logic. The smaller time period is typical for microprogrammed machines, this particular value (and other information on microprogramming) being taken from Varian Data's 620/i model [Ref. 7]. Also, the micro-instruction cycle time (784 nsec.) is reasonable for the DMS2 Controller model.

The simulation results in Appendix C demonstrate that a microprogrammed device can implement the commands of a conventional controller. The next chapter shows that the microprogrammed control concept can be extended to much more complex functions.

V. SMART MICROPROGRAMMED DISC CONTROLLER

The smart microprogrammed disc controller described in this chapter is a powerful extension of the "dumb" microprogrammed version. It is able to handle both the basic functions required of a disc controller and certain data handling functions as well. These data-manipulation functions do operations which would otherwise be handled by the CPU. Thus, the smart disc controller frees the main computer from some of its tasks and maintains the advantages of microprogramming as well.

A. FUNCTIONS AND THEIR EXECUTIONS

1. Basic Functions

The six basic functions executed by the two previous models are handled by the smart controller also. It executes these exactly as the dumb controller, and, because of its more powerful capabilities, it extends the basic function set to include other similar but useful operations.

One example of these new functions is the Write Format command. Its machine code is completely equivalent to those of the previous models with one exception. There is, in addition to the data specified in previous codes, a two-bit operation code. This code causes the controller to do one of three things: 1.) write header information on all sectors within the current cylinder, or 2.) write header information from cylinder zero up to the specified cylinder, or 3.) write

header information from the specified first cylinder to the specified last cylinder. In effect, this command combines the function of the Write Format and Seek commands to form a loop, then makes use of its extra registers and ALU to execute it.

Similar other extensions to the basic functions set have been added by combining functions. For example, the Seek and Select commands are combined to allow the operator to specify both the drive and cylinder at the same time. Furthermore, the Read and Write commands can be improved upon extensively. It is often desirable to read from or write on not only a sector, but a specific portion of the sector; this is done in the smart controller. A closer look at that aspect will be taken in the following section.

2. Possible Extended Functions

The primary goal of the smart microprogrammed controller is to relieve the central computer of some of its load by assimilating some of the computing. This goal is achieved by the introduction of extended functions. These functions are practical due to the development of microprogramming and the demand for methods to reduce computing time of the CPU.

One use of extended functions is in file maintenance. In the system under analysis, many records of a file may be kept within a sector, which is the smallest addressable unit; however, it is desirable to address and manipulate these records individually. With a sixteen-bit command word, one

may address up to sixteen records per sector. Moreover, a smart controller, because of its extra registers, can find a record within a sector without a need for marks at the beginning of each one. A record could consist of six thirty-two bit words: the key, a forward pointer, backwards pointer, and three words of text. The key would contain cylinder, track, sector and record information for verification purposes, while the pointers would help in file manipulation.

As an example of how the pointers may be used, the function Insert can be analyzed. The Insert command would read from the CPU the text of the record to be inserted (three words) and temporarily store it into its registers. The text would be analyzed and its proper position in the file would be determined by some criterion (alphabetically, perhaps). If it was decided that the new record should be between records A and B, the following sequence would occur. First, an available space near A or B for the new record would be found. In that space would be written the key, the forward pointer (location of B), backwards pointer (location of A), and the three-word text. Finally the computer would change A's forward pointer and B's backwards pointer to be the position of the new record.

Another use of a smart controller is to have it control the available space file. When a new record needs to be inserted, as in the above example, the controller

searches the list of available spaces to locate a convenient space. Likewise, upon deletion of a record, the now-available space is added in order to update the list.

Similar to the control of available space is garbage collection, which, although more involved, could be handled by a smart controller. Implementation of other powerful functions, such as changing a certain parameter within each record of an entire list, or rearrangement of the list (e.g. sorting, merging, reordering, etc.) is certainly feasible.

B. STRUCTURE

The availability of extended functions is due in part to the advantages of microprogarmming. More important, however, are the additions to the dumb controller which distinguish the two versions and allows one to refer to this model as a smart controller. The new portions are represented in block diagram form in Figure 5. Blocks having the same labels as in earlier models are identical to them. The additions are as follows:

ALU — Arithmetic-Logic Unit. Handles all arithmetic functions, as well as incrementing, Boolean operations, etc.

SP1-SP4 — Scratch Pad registers which the ALU uses for its computations and temporary storage.

S-Bus — Communication link between ALU and Scratch Pads.

Extended Data Buffer — Contains extra registers in order to hold more data for input or output than was buffered in the conventional or dumb microprogrammed controllers.

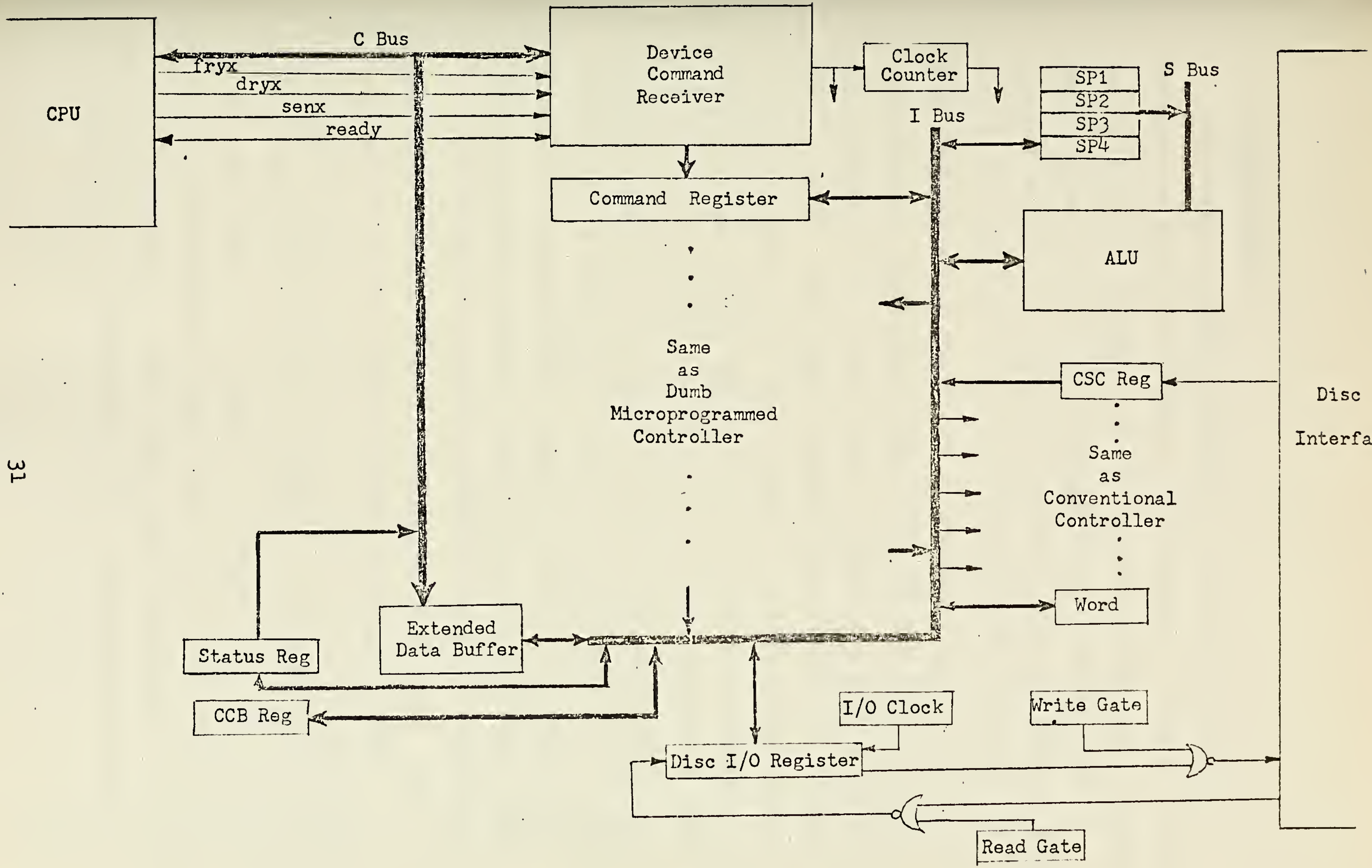

CPU
C Bus
fryx
dryx
senx
ready
Device Command Receiver
Clock Counter
SP1
SP2
SP3
SP4
S Bus
I Bus
Command Register
ALU
Same as Dumb Microprogrammed Controller
CSC Reg
Same as Conventional Controller
Word
Disc Interfa
Extended Data Buffer
Status Reg
CCB Reg
I/O Clock
Write Gate
Disc I/O Register
Read Gate

Figure 5.

C. FURTHER WORK

The value of a smart controller lies in its ability to relieve the CPU of its repetitious operations. Thus, it would be advisable to attempt to apply the smart controller in areas which currently require a large amount of data shuffling operations, such as those found in data base management. Much computer time could be saved if peripheral devices could be made to handle this "busy work," that is, computer operations which require transferring data from file storage (disc) to main memory (core), making small changes, and moving the data back to disc. The examples mentioned previously as extended functions were not very complicated, but powerful and esoteric microprograms are certainly within the realm of the smart controller's ability. If microprograms requiring more computational ability are desired, this may require addition of data-handling registers. However, modification of the logic would be straightforward, since the ROM is not complicated by the extensive timing circuits found in conventional logic.

Development of extended functions and their simulation should be the next step in the study of this topic. The simulation could be similar to that done for the dumb microprogrammed controller, and would require the development of a new function set and, perhaps, a different microinstruction word length. One desired outcome of a careful simulation would be an estimate of the amount of CPU time saved by one type of peripheral device over another when similar operations are executed. An important part of the simulation would be

the computation of the times required using the different controllers to accomplish the same data processing tasks. In the smart controller one must simply count the number of microinstruction cycles and multiply by the time per cycle, whereas in the conventional controller an average time could be assigned to each command. In both cases the total activity to perform tasks involving hundreds of device commands must be calculated.

Example application areas to be simulated should include the updating or maintenance of a large data base such as an inventory of equipment or stores, or a personnel file. Simulating file operations in such an application would identify how much channel activity, main storage occupancy, and CPU activity can be saved by the implementation of certain extended functions. Also, these simulations would identify which possible extended functions are likely to have the highest payoff. Another area of development could be the redesign of the controller based on a system other than the ADAGE. In any case, further possibilities of such a system certainly merit investigation.

VI. SUMMARY AND CONCLUSIONS

The principles of disc drive controllers and microprogramming were presented. For ease of analysis, a conventional controller model based on an existing system was then developed and its operation was outlined. Next, the logic of the model was replaced by microprogrammed logic which handled the same functions. Finally, the microprogrammed model was extended to form a smart disc controller which could handle more computational operations. The three models were then compared, and it was pointed out how the smart version could save computer time due to its capability of handling more powerful functions. Also, development of such functions and their simulation were recommended as directions for future study.

From the investigation of the topic, certain conclusions are evident. First, due to its ease of modification and design, microprogramming can be advantageously applied to peripheral controllers. Also, controllers which would carry out more powerful functions would be more practical since they could help alleviate the workload of the central computer and I/O channels. Although devices could be developed to handle extensive computation, smart controllers would be most beneficial in applications involving much data shuffling and small computation, such as file maintenance and related areas.

APPENDIX A

COMPLETE MICROPROGRAM FOR DUMB CONTROLLER

Address	I-Bus & Special	Set Reset	Function	Store	Skip
SELECT DRIVE					
0006	NOP	BIT	NOP	NOP	NOP
0007	C2	NOP	NOP	DE	EOP
SEEK					
0010	NOP	D	CR		
0011	C1	NOP	SUB	A	NOP
0012	OPP	STP	NOP	NOP	NFG
0013	NOP	NOP	JMP	[20]	
0014	A	NOP	RPT	CSC	NOP
0015	CSC	NOP	NOP	A	AZR
0016	NOP	SL01	CR	[020]	
0017	A	NOP	RPT	CSC	UNC
0020	C1	SL01	RPT	CSC	NOP
0021	CSC	NOP	NOP	A	AZR
0022	NOP	STP1	NOP	NOP	EOP
WRITE FORMAT					
0023	NOP	BIT	RPT	NOP	NOP
0024	WIP	CCB	NOP	A	ODD
0025	NOP	WIP	NOP	NOP	NOP
0026	*BIT	BITK	CR,RPT	[-3]	
0027	BIT	NOP	NOP	A	AZR
0030	*WD	WC	CR,RPT	[-2]	
0031	WD	NOP	NOP	A	AZR
0032	*BIT	IOK	CR,RPT	[-31]	
0033	BIT	NOP	NOP	A	AZR
0034	*BIT	IOO	CR,RPT	[-8]	
0035	BIT	NOP	NOP	A	AZR
0036	TRACK	NOP	RPT	I/O	NOP

Address	I-Bus & Special	Set Reset	Function	Store	Skip
0037	BYTE	NOP	NOP	A	ODD
0040	CAR	NOP	NOP	I/O	NOP
0041	*BIT	NOP	CR,RPT	[-8]	
0042	BIT	NOP	NOP	A	AZR
0043	SECT	NOP	RPT	I/O	NOP
0044	BYTE	NOP	NOP	A	ODD
0045	CCB	NOP	RPT	I/O	NOP
0046	BYTE	NOP	NOP	A	AZR
0047	*WD	BIT	CR,RPT	[-2]	
0050	WD	NOP	NOP	A	AZR
0051	*BIT	NOP	CR,RPT	[-31]	
0052	BIT	NOP	NOP	A	AZR
0053	NOP	IOO	RPT	NOP	NOP
0054	WSP	NOP	NOP	A	ODD
0055	SECT	WSP	NOP	A	AZR
0056	NOP	BIT	JMP	[026]	
0057	NOP	IOK,WG	NOP	NOP	NOP
0060	NOP	BITK	NOP	NOP	EOP
READ					
0061	NOP	RG	JSB	[0106]	
0062	*BIT	BIT	CR,RPT	[-8]	
0063	BIT	NOP	NOP	A	AZR
0064	I/O	BIT	NOP	DB	NOP
0065	RDY	NOP	NOP	A	ODD
0066	NOP	NOP	JMP	[071]	
0067	DB,C	NOP	NOP	CPU	NOP
0070	NOP	NOP	JMP	[062]	
0071	NOP	IOK	NOP	NOP	NOP
0072	NOP	IODB	NOP	NOP	EOP
WRITE					
0073	CPU,C	RG	NOP	DB	NOP
0074	NOP	NOP	JSB	[0106]	
0075	DB	WG,RG	NOP	I/O	NOP

Address					
0076	CPU,C	NOP	RPT	DB	NOP
0077	BYTE	NOP	NOP		ODD
0100	DB	BIT	NOP	I/O	NOP
0101	RDY	NOP	NOP	A	AZR
0102	NOP	NOP	JMP	[076]	
0103	NOP	NOP	RPT	NOP	NOP
0104	BYTE	NOP	NOP	A	ODD
0105	NOP	IOK,WG	NOP	NOP	EOP

<u>READ OR WRITE</u>

0106	NOP	IOK	RPT	NOP	NOP
0107	WSP	NOP	RPT	A	ODD
0110	I/O	WSP	NOP	A	NEG
0111	C3	BIT	NOP	A	AZR
0112	NOP	BITK	NOP	NOP	NOP
0113	*BIT	TRACK	CR,RPT	[-16]	
0114	BIT	NOP	NOP	A	AZR
0115	*BIT	NOP	CR	[-8]	
0116	I/O	NOP	NOP	A	NOP
0117	CAR	NOP	SUB	A	NOP
0120	NOP	NOP	NOP	NOP	AZR
0121	NOP	NOP	JMP	[0106]	
0122	NOP	NOP	RPT	NOP	NOP
0123	BYTE	NOP	NOP	A	ODD
0124	I/O	NOP	NOP	A	NOP
0125	C2	NOP	SUB	A	NOP
0126	NOP	NOP	NOP	NOP	AZR
0127	NOP	NOP	JMP	[0106]	
0130	NOP	NOP	RPT	NOP	NOP
0131	SYNC	NOP	NOP	A	ODD
0132	NOP	BIT	RSB	NOP	NOP

APPENDIX B

MICROINSTRUCTION FIELDS

I-Bus and Special	Set-Reset	Function	Store	Skip

I-Bus and Special — Places contents of specified register on I-Bus or performs a special function:

A — A Register
BIT — BIT Register
BYTE — BYTE Register
C1 — Command Register (8-15)
C2 — Command Register (4-7)
C3 — Command Register (5)
CAR — Cylinder Address Register
CPU — CPU's Data Register
CSC — Cylinder Step Counter
CCB — Cyclic Check Byte Register
DB — Data Buffer
I/O — I/O Register
RDY — Ready Line
SECT — Sector Register
SYNC — Syncbit Line
TRACK — Track FF
WIP — Wait for Index Pulse FF
WSP — Wait for Sector Pulse FF
WORD — Word Register
OPP — Causes program to skip next statement if conditions in the skip field are not met.
*BIT,*WORD — Allows storing a specified constant in BIT or WORD Register.
C — When a C appears with another code, contents are to be placed on C-bus vice I-bus.
NOP — No Operation.

Set-Reset – Causes a pulse to be transmitted to the appropriate flip-flop or register:

BIT — Resets BIT, BYTE, and WORD Registers
BITK — Controls clock to BIT Register
CCB — Resets Cyclic Check Byte
D — Change Direction Flip-Flop
IO0 — Sets bit zero of I/O Register
IODB — Resets I/O and Data Buffer Registers
IOK — Controls clock to I/O Register
RG — Controls Read Gate
SL01 — Sets Seek Slow Flip-Flop
SL02 — Resets Seek Slow Flip-Flop and sets Seek Stop Flip Flop
STP — Change Seek Stop Flip-Flop
TRACK — Change Track Flip-Flop
WG — Controls Write Gate
WIP — Resets Wait for Index Pulse Flip-Flop
WSP — Resets Wait for Sector Pulse Flip-Flop

Note: In some cases, more than one code may appear in the field.

Function — Handles data manipulation anc changes in normal addressing sequence.

CR — A constant (specified in bits 0-7) is placed on I-Bus and stored in A Register, unless *BIT or *WORD appear in I-Bus Field.

JMP — Causes next addressed microinstruction to be that specified in bits 0-7.

JSB — Same as JMP, with return address (that of next microinstruction) stored in Save Register.

RSB — Next addressed instruction is that stored in Save Register

RPT — Sets Repeat mode for next microinstruction , causing next instruction to be repeated until conditions in Skip Field are met. RPT may appear with another code (e.g. CR,RPT).

SUB — Contents of I-Bus subtracted from A-Register; results placed in A-Register.

Store Field — Contents of I-Bus sent to specified destination.

A	— A Register
CPU	— CPU Data Register
CSC	— Cylinder Step Counter
DB	— Data Buffer
DE	— Drive Enable Lines
I/O	— I/O Register

Skip Field — Cause program to skip next microinstruction if conditions specified in this one are met.

AZR	— Skips if $A = 0$
NEG	— Skips if $A < 0$
ODD	— Skips if $A(0) = 1$
UNC	— Skips unconditionally
EOP	— Resets controller-busy bit in Status Register. Used in next to last microinstruction in each command.

APPENDIX C

OUTPUT OF SIMULATION PROGRAM

```
=================================== P A G E   S K I P   S U P P R E S S E D =========================

     SEEK  CYLINDER - - 0010000101100100

     ROM
   ADDRESS              MICROINSTRUCTION                       REGISTERS AND FLIP-FLOPS

                                                    D FF  A REG.  SKSLOW  SKSTOP  CSC  CAR
                                                     0      0       0       1      0    0
   ** 0010  | NOP .   D .  CR .    20      |
                                                     1     20       0       1      0    0
   ** 0011  |  C1 .  NOP . SUB .    A . NOP |
                                                     1     80       0       1      0    0
   ** 0012  | OPP .  STP . NOP . NOP . NEG |
                                                     1     80       0       0      0    0
   ** 0014  |   A .  NOP . RPT . CSC . NOP |
                                                     1     80       0       0     80    0
   ** 0015  | CSC .  NOP . NOP .    A . AZR |
                                                              . . . .
                                                     1      0       0       0      0   80
   ** 0016  | NOP . SL01 .  CR .    20     |
                                                     1     20       1       0      0   80
   ** 0017  |   A .  NOP . RPT . CSC . UNC |
                                                     1     20       1       0     20   80
   ** 0021  | CSC .  NOP . NOP .    A . AZR |
                                                     1      0       1       0      0  100
   ** 0022  | NOP . STP1 . NOP . NOP . EOP |
                                                     1      0       0       1      0  100
```

REFERENCES

1. ADAGE Hardware Maintenance Manual, Vol. I, "Disk Memory Subsystem, DMS2," 1-B15, September, 1968.

2. Hewlett-Packard Co., Microprogramming Guide, for HP Model 2100 Computer, November, 1971.

3. Husson, Samir S., Microprogramming Principles and Practices, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1970.

4. Knuth, Donald E., Fundamental Algorithms - The Art of Computer Programming, p. 406-420, Addison-Wesley, 1968.

5. Stone, Harold S., Introduction to Computer Organization and Data Structures, p. 203-218, 247-280, McGraw-Hill, 1972.

6. Williams, Robin, "A Survey of Data Structures for Computer Graphics Systems," Computing Surveys, v. 3, p. 1-17, March, 1971.

7. Varian Data Machines, Varian Data 620/i Computer Manual, 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Asst. Prof. V. Michael Powers Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	ENS. Edwin Ladeau Tomlin, Jr. 3212 Hudson Apt. 19 Tulsa, Oklahoma 74135	1

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Microprogrammed Disc Controllers		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1973
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edwin Ladeau Tomlin, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1973
		13. NUMBER OF PAGES 45
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprogramming
Disc Controller
Smart Controller

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Disc controllers, the concepts of microprogramming, and the combination of the two are presented. First, the conventional controller is discussed and a model developed. Logic for the model is then replaced by microprogramming to form a new model. Finally, the new version is modified to handle extended logic. The functions, structure, and relative merits of the three models are discussed, as well as suggestions for further work.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DD Form 1473 (BACK)
1 Jan 73
S/N 0102-014-6601

Thesis
T717 Tomlin 140846
c.1 Microprogrammed disc
controllers.

Thesis 140846
T717 Tomlin
c.1 Microprogrammed disc
controllers.

thesT717
Microprogrammed disc controllers.
3 2768 001 01127 3
DUDLEY KNOX LIBRARY